Snapshot Waveforms

Access to Data Streams

Mon, May 15, 2000

In order to buffer up waveforms so that a console application can collect 15Hz waveforms, the data stream mechanism is used. A simple approach is to allocate data stream queues that are large enough to hold 2–3 waveforms. A RETDAT request made for 7.5Hz replies can then provide the required buffering to keep up with 15Hz, even though the console application executes at only an approximate and asynchronous 15Hz rate. See the document called Reliable Collection of 15Hz Data for more background on the problem.

It is desired to support access to these waveforms both via RETDAT and FTPMAN. One can use FTPMAN for obtaining a single snapshot. One must use RETDAT for obtaining 15Hz snapshots.

Using RETDAT, a console can access the 15Hz waveforms by using an array device that reads from the data stream queue into a buffer that is at least large enough for two data stream records, plus a 4-byte header. This implies that the application knows the size of a record. The record size will be fixed as part of the data stream definition in the DSTRM table.

A new piece of logic added to FTPMAN can support access to these data stream waveforms. A new CINFO entry is called for to support this. It must include:

Channel#
Data stream#
Offset to actual waveform data in data stream record

What information must be saved with the waveform in order to identify it well enough to build an FTPMAN reply? The snapshot parameters are usually clock event#, delay, rate, and the number of points digitized. These will be used for the status reply. A time-stamp may be important, but there is no way to return the time-stamp using the FTPMAN Snapshot protocol. But since FTPMAN does not operate at 15Hz, the time-stamp is not necessary. The console can do its own time-stamping.

FTPMAN responds with a status update reply that lets the client know when the snapshot data is ready. We should wait for a new record to be written into the data stream after the request has been initialized. Then we can capture the waveform by copying the most recent one from the data stream queue and declare that the data is available and that the console may "come and get it."

What logic will copy a waveform record into the data stream? There is a data access table entry type that does something like this, but we need more logic than it supplies. A local application called WFDS might do this job. Its parameters would include a reference to the waveform memory, a step size, and an initial data stream number. On any Booster reset, or on any of those indicated by a parameter, we can copy each waveform into a separate data stream. For the specific case of Booster beam loss monitors, it may be good to include the data stream recording along with the rest of the

required logic. In this case, a function can supply the waveform copying logic. It can also perform the linearization and average dose calculations over specified time periods. The name of an LA for the Booster BLM case might be BLMS.

What will be saved as a part of the waveform record header? The time-of-day should be included. An indication of which waveform (memory address) and the Booster reset event number can be included. The contents of this header must be known to an application that reads from the data stream. We may not need the waveform if each data stream uses a separate waveform. A 16-byte header might be sufficient.

Field	*Size*
Waveform address	4
Booster reset event	1–2
Time-of-day	8

In order for FTPMAN to find the parameter values associated with a waveform it obtains from the data stream, it must know something about the header. Should the header include these parameters? If not, they would have to be found in the channels that hold them whose analog control fields point to the appropriate registers. It seems better if they can be part of the data record, since they influenced the waveform that was sampled. If an event did not occur for awhile, the data stream may not be refreshed often, so its contents may not match the current values of the parameter channels.

FTPMAN could call a function that returns a pointer to the next record, given a pointer to the previous record. If the input pointer is NIL, a pointer to the next record to be written is returned. If that pointer is passed to the function right away, the NIL pointer is returned. If NIL is passed, and NIL is returned, there is an error.

```
DSNext(dsNum: Integer; dsRecP: DSRecordPtr): DSRecordPtr;
```